8. NumPy & Pandas

Numpy란?

Numpy는 C언어로 구현된 파이썬 라이브러리로써, 고성능의 수치계산을 위해 제작되었다. Numerical Python의 줄임말 이기도 한 Numpy는 벡터 및 행렬 연산에 있어서 매우 편리한 기능을 제공한다.

또한 이는 데이터분석을 할 때 사용되는 라이브러리인 pandas와 matplotlib의 기반으로 사용되기도 한다.

numpy에서는 기본적으로 array라는 단위로 데이터를 관리하며 이에 대해 연산을 수행한다. array는 말그대로 행렬이라는 개념으로 생각하면 된다.

먼저 numpy를 사용하기 위해서는 아래와 같은 코드로 numpy를 import해야 한다.

```
import numpy as np
```

사실상, numpy를 설치하고 단순히 import numpy 만 해도 되지만, 이를 코드에서 보다 편하게 사용하기 위해 as np 를 붙임으로써 np라는 이름으로 numpy를 사용한다.

Array 정의 및 사용하기

```
import numpy as np

list1 = [1, 2, 3, 4]
a = np.array(list1)
print(a.shape)

b = np.array([[1,2,3],[4,5,6]])
print(b.shape)
print(b[0,0])
```

```
(4,)
(2, 3)
1
```

numpy shape

numpy에서는 해당 array의 크기를 알 수 있다.
shape 을 확인함으로써 몇개의 데이터가 있는지, 몇 차원으로 존재하는지 등을 확인할 수 있다.
위에서 a.shape의 결과는 (4,) 으로써, 1차원의 데이터이며 총 4라는 크기를 갖고 있음을 알 수 있다.
b.shape의 결과는 (2,3) 으로써, 2차원의 데이터이며 2 * 3 크기를 갖고 있는 array 이다.

numpy 자료형

- 부호가 있는 정수 int(8, 16, 32, 64)
- 부호가 없는 정수 uint(8 ,16, 32, 54)
- 실수 float(16, 32, 64, 128)
- 복소수 complex(64, 128, 256)
- 불리언 bool
- 문자열 string_
- 파이썬 오프젝트 object
- 유니코드 unicode_

Array 정의 및 사용하기


```
import numpy as np

a = np.zeros((2,2))
print(a)

a = np.ones((2,3))
print(a)

a = np.full((2,3), 5)
print(a)

a = np.eye(3)
print(a)

a = np.array(range(20)).reshape((4,5))
print(a)
```

```
[[0. 0.]
 [0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]]
[[5 5 5]
 [5 5 5]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

np.arange() 함수는 인자로 받는 값 만큼 1씩 증가하는 1차원 array를 만든다.
이때 하나의 인자만 입력하면 0 ~ 입력한 인자, 값 만큼의 크기를 가진다.

```
import numpy as np

lst = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
arr = np.array(lst)

a = arr[0:2, 0:2]
print(a)

a = arr[1:, 1:]
print(a)
```

```
[[1 2]
 [4 5]]
[[5 6]
 [8 9]]
```

NumPy 정수 인덱싱

```
import numpy as np

lst = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12]
]
a = np.array(lst)

s = a[[0, 2], [1, 3]]

print(s)
```

```
[ 2 12]
```

numpy에서 사용되는 인덱싱은 기본적으로 python 인덱싱과 동일하다.

이때, python에서와 같이 1번째로 시작하는 것이 아니라 0번째로 시작하는 것에 주의한다.

NumPy 부울린 인덱싱


```
import numpy as np

lst = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
a = np.array(lst)

bool_indexing_array = np.array([
    [False,  True, False],
    [True, False,  True],
    [False,  True, False]
])

n = a[bool_indexing_array];
print(n)
```

```
[2 4 6 8]
```

위에서 이용한 다차원의 인덱싱을 응용하여 boolean 인덱싱을 할 수 있다.

해당 기능은 주로 마스크라고 이야기하는데, boolean인덱싱을 통해 만들어낸 array를 통해

우리가 원하는 행 또는 열의 값만 뽑아낼 수 있다.

즉, 마스크처럼 우리가 가리고 싶은 부분은 가리고, 원하는 요소만 꺼낼 수 있다.

NumPy 부울린 인덱싱

```
import numpy as np

lst = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
a = np.array(lst)

bool_indexing = (a % 2 == 0)
print(bool_indexing)
print(a[bool_indexing])

n = a[ a % 2 == 0 ]
print(n)
```

```
[[False  True False]
 [ True False  True]
 [False  True False]]
[2 4 6 8]
[2 4 6 8]
```

8. NumPy & Pandas

8.1 NumPy I

8.2 NumPy II

8.3 Pandas I

8.4 Pandas II

NumPy 연산

```
import numpy as np

a = np.array([1,2,3])
b = np.array([4,5,6])

c = a + b
print(c)

c = a - b
print(c)

c = np.multiply(a, b)
print(c)

c = np.divide(a, b)
print(c)
```

```
[5 7 9]
[-3 -3 -3]
[ 4 10 18]
[0.25 0.4  0.5 ]
```

기본적으로 numpy에서 연산을 할때는 크기가 서로 동일한 array 끼리 연산이 진행된다.

이때 같은 위치에 있는 요소들 끼리 연산이 진행된다.

행렬의 곱처럼 곱셈이 진행되는 것이 아니라 각 요소별로 곱셈이 진행된다.

브로드 캐스팅


```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)

for i in range(4):
    y[i, :] = x[i, :] + v

print(y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

앞 에서는 array가 같은 크기를 가져야 서로 연산이 가능하다고 했지만,

numpy에서는 브로드캐스트라는 기능을 제공한다.

브로드캐스트란, 서로 크기가 다른 array가 연산이 가능하게끔 하는 것이다.

브로드 캐스팅

```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))
print(vv)
y = x + vv
print(y)
```

```
[[1 0 1]
 [1 0 1]
 [1 0 1]
 [1 0 1]]
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

np.tile(v,(4,1)) 은 v의 복사본 4개를 차곡차곡 쌓는다.

브로드 캐스팅


```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v
print(y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

x의 shape가 (4, 3)이고 v의 shape가 (3,)라도 브로드캐스팅으로 인해 y = x + v는 문제없이 수행된다. 이때 'v'는 'v'의 복사본이 차곡차곡 쌓인 shape (4, 3)처럼 간주되어 'x'와 동일한 shape가 되며 이들 간의 요소별 덧셈연산이 y에 저장된다.

브로드 캐스팅


```
import numpy as np

v = np.array([1,2,3])
w = np.array([4,5])
print(np.reshape(v, (3, 1)) * w)

x = np.array([[1,2,3], [4,5,6]])
print(x + v)

print((x.T + w).T)
print(x + np.reshape(w, (2, 1)))

print(x * 2)
```

```
[[ 4  5]
 [ 8 10]
 [12 15]]
[[2 4 6]
 [5 7 9]]
[[ 5  6  7]
 [ 9 10 11]]
[[ 5  6  7]
 [ 9 10 11]]
[[ 2  4  6]
 [ 8 10 12]]
```

행렬 곱 연산


```
import numpy as np

lst1 = [
    [1,2],
    [3,4]
]

lst2 = [
    [5,6],
    [7,8]
]
a = np.array(lst1)
b = np.array(lst2)

c = np.dot(a, b)
print(c)
```

```
[[19 22]
 [43 50]]
```

행렬 전치


```
import numpy as np

x = np.array([[1,2], [3,4]])
print(x)

print(x.T)


v = np.array([1,2,3])
print(v)
print(v.T)
```

```
[[1 2]
 [3 4]]
[[1 3]
 [2 4]]
[1 2 3]
[1 2 3]
```

행렬 열간, 행간 통계

```
import numpy as np

a = np.array([[1,2],[3,4]])

s = np.sum(a)
print(s)

s = np.sum(a, axis=0)
print(s)

s = np.sum(a, axis=1)
print(s)

s = np.prod(a)
print(s)
```

```
10
[4 6]
[3 7]
24
```

8. NumPy & Pandas

8.1 NumPy I

8.2 NumPy II

8.3 Pandas I

8.4 Pandas II

데이터 프레임 만들기 1


```
import pandas as pd

dict_data = {'c0':[1,2,3,4,5], 'c1':[6,7,8,9,10]}
df1 = pd.DataFrame(dict_data)
print(type(df1), "\n")
print(df1, "\n")

df2 = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2', 'r3', 'r4'])
print(df2, "\n")

list_of_list_data = [[1,2,3,4,5], [6,7,8,9,10]]
df3 = pd.DataFrame(list_of_list_data)
print(df3, "\n")

df4 = pd.DataFrame(list_of_list_data,
                   index=['r0', 'r1'],
                   columns=['c0', 'c1', 'c2', 'c3', 'c4'])
print(df4, "\n")

print(df2.shape)
print(df4.shape)
```

데이터 프레임 만들기 1

```
<class 'pandas.core.frame.DataFrame'>

   c0  c1
0   1   6
1   2   7
2   3   8
3   4   9
4   5  10

    c0  c1
r0   1   6
r1   2   7
r2   3   8
r3   4   9
r4   5  10

   0  1  2  3   4
0  1  2  3  4   5
1  6  7  8  9  10

    c0  c1  c2  c3  c4
r0   1   2   3   4   5
r1   6   7   8   9  10

(5, 2)
(2, 5)
```

데이터 프레임 만들기 2

```
import pandas as pd

csv_file = 'bok_statistics_CD.csv'

df1 = pd.read_csv(csv_file)
print(df1)
print('\n')

df2 = pd.read_csv(csv_file, header=None)
print(df2)
print('\n')

df3 = pd.read_csv(csv_file, index_col=0)
print(df3)
print('\n')

df4 = pd.read_csv(csv_file, index_col=0, header=None)
print(df4)
print('\n')

excel_file = 'report_Key100Stat.xls'

df5 = pd.read_excel(excel_file)
print(df5)
```

데이터 프레임 만들기 2

```
          2013   2.72   -0.58
0         2014   2.49   -0.23
1         2015   1.76   -0.73
2         2016   1.49   -0.27
3         2017   1.44   -0.05
4         2018   1.68    0.24
5         2019   1.69    0.01
6      2019.10   1.46   -0.08
7      2019.11   1.52    0.06
8      2019.12   1.53    0.01
9       2020.1   1.47   -0.06
10      2020.2   1.42   -0.05
11      2020.3   1.23   -0.19
12      2020.4   1.10   -0.13
13  2020.5.22   1.02    0.00
14  2020.5.25   1.02    0.00
15  2020.5.26   1.02    0.00
16  2020.5.27   1.02    0.00
17  2020.5.28   0.81   -0.21
18  2020.5.29   0.81    0.00
...
```

데이터프레임의 내용 살펴보기


```
import pandas as pd

df = pd.read_csv('bok_statistics_CD.csv', header=None)

print(df.head())
print('\n')
print(df.head(3))
print('\n')
print(df.tail())
print('\n')
print(df.tail(3))
```

데이터프레임의 내용 살펴보기

```
      0     1     2
0  2013  2.72 -0.58
1  2014  2.49 -0.23
2  2015  1.76 -0.73
3  2016  1.49 -0.27
4  2017  1.44 -0.05

      0     1     2
0  2013  2.72 -0.58
1  2014  2.49 -0.23
2  2015  1.76 -0.73

             0     1     2
15  2020.5.25  1.02  0.00
16  2020.5.26  1.02  0.00
17  2020.5.27  1.02  0.00
18  2020.5.28  0.81 -0.21
19  2020.5.29  0.81  0.00

             0     1     2
17  2020.5.27  1.02  0.00
18  2020.5.28  0.81 -0.21
19  2020.5.29  0.81  0.00
```

데이터프레임의 요약정보 확인하기 1


```
import pandas as pd

df = pd.read_csv('bok_statistics_CD.csv', header=None)

print(df.head())
print('\n')
print(df.info())
```

데이터프레임의 요약정보 확인하기 1

```
      0     1     2
0  2013  2.72 -0.58
1  2014  2.49 -0.23
2  2015  1.76 -0.73
3  2016  1.49 -0.27
4  2017  1.44 -0.05


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       20 non-null     object
 1   1       20 non-null     float64
 2   2       20 non-null     float64
dtypes: float64(2), object(1)
memory usage: 608.0+ bytes
None
```

데이터프레임의 요약정보 확인하기 2

```
import pandas as pd

df = pd.read_csv('bok_statistics_CD.csv', header=None)

print(df.head())
print('\n')
print(df.describe())
```

데이터프레임의 요약정보 확인하기 2

```
      0     1     2
0  2013  2.72 -0.58
1  2014  2.49 -0.23
2  2015  1.76 -0.73
3  2016  1.49 -0.27
4  2017  1.44 -0.05


                1          2
count  20.000000  20.000000
mean    1.435000  -0.113000
std     0.494214   0.219236
min     0.810000  -0.730000
25%     1.020000  -0.195000
50%     1.450000  -0.050000
75%     1.567500   0.000000
max     2.720000   0.240000
```

8. NumPy & Pandas

8.1 NumPy I

8.2 NumPy II

8.3 Pandas I

8.4 Pandas II

데이터프레임의 열 다루기

```
import pandas as pd

dict_data = {'c0':[1,2,3], 'c1':[4, 5, 6]}
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df, "\n")

col0 = df['c0']
col1 = df.c1

print(col0, "\n")
print(col1, "\n")

df['c2'] = 7, 8, 9
print(df, "\n")

df['c3'] = 0
print(df, "\n")
```

데이터프레임의 열 다루기

```
df['c4'] = df['c3']
print(df, "\n")

df['c3'] = 10, 11, 12
print(df, "\n")

df['c3'] = 0
print(df, "\n")

df.drop('c4', axis=1, inplace=True)
print(df, "\n")

df.drop(['c1','c3'], axis=1, inplace=True)
print(df)
```

데이터프레임의 열 다루기

```
    c0  c1
r0   1   4
r1   2   5
r2   3   6

r0    1
r1    2
r2    3
Name: c0, dtype: int64

r0    4
r1    5
r2    6
Name: c1, dtype: int64

    c0  c1  c2
r0   1   4   7
r1   2   5   8
r2   3   6   9

    c0  c1  c2  c3
r0   1   4   7   0
r1   2   5   8   0
r2   3   6   9   0
```

데이터프레임의 열 다루기

```
    c0  c1  c2  c3  c4
r0   1   4   7   0   0
r1   2   5   8   0   0
r2   3   6   9   0   0

    c0  c1  c2  c3  c4
r0   1   4   7  10   0
r1   2   5   8  11   0
r2   3   6   9  12   0

    c0  c1  c2  c3  c4
r0   1   4   7   0   0
r1   2   5   8   0   0
r2   3   6   9   0   0

    c0  c1  c2  c3
r0   1   4   7   0
r1   2   5   8   0
r2   3   6   9   0

    c0  c2
r0   1   7
r1   2   8
r2   3   9
```

데이터프레임의 행 다루기

```
import pandas as pd

dict_data = {'c0':[1,2,3], 'c1':[4, 5, 6]}
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df, "\n")

row0 = df.iloc[0]
row1 = df.iloc[1]
row2 = df.loc['r2']

print(row0, "\n")
print(row1, "\n")
print(row2, "\n")

df.loc['r3'] = 10, 20
print(df, "\n")
```

```
df.loc['r4'] = 0
print(df, "\n")

df.loc['r3'] = df.loc['r4']
print(df, "\n")

df.drop('r4', axis=0, inplace=True)
print(df, "\n")

df.drop(['r1','r3'], axis=0, inplace=True)
print(df)
```

데이터프레임의 행 다루기

```
    c0  c1
r0   1   4
r1   2   5
r2   3   6

c0    1
c1    4
Name: r0, dtype: int64

c0    2
c1    5
Name: r1, dtype: int64

c0    3
c1    6
Name: r2, dtype: int64

    c0  c1
r0   1   4
r1   2   5
r2   3   6
r3  10  20
```

데이터프레임의 행 다루기

```
    c0  c1
r0   1   4
r1   2   5
r2   3   6
r3  10  20
r4   0   0

    c0  c1
r0   1   4
r1   2   5
r2   3   6
r3   0   0
r4   0   0

    c0  c1
r0   1   4
r1   2   5
r2   3   6
r3   0   0

    c0  c1
r0   1   4
r2   3   6
```

```
import pandas as pd

dict_data = {'c0':[1,2,3], 'c1':[4, 5, 6]}
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df, "\n")

el_01 = df.iloc[0,1]
print(el_01, "\n")

el_11 = df.iloc[1,1]
print(el_11, "\n")

el_21 = df.loc['r2','c1']
print(el_21, "\n")

el_12_01 = df.loc['r1':'r2','c0':'c1']
print(el_12_01, "\n")
```

```
el_12_01_iloc = df.iloc[1:3, 0:2]
print(el_12_01, "\n")

df.iloc[0,1] = 40
print(df, "\n")

df.iloc[1:3, 0:2] = 0
print(df)
```

```
    c0  c1
r0   1   4
r1   2   5
r2   3   6

4

5

6

    c0  c1
r1   2   5
r2   3   6

    c0  c1
r1   2   5
r2   3   6
```

```
    c0  c1
r0   1  40
r1   2   5
r2   3   6

    c0  c1
r0   1  40
r1   0   0
r2   0   0
```

데이터프레임을 외부 파일로 저장하기

```
import pandas as pd

dict_data = {'c0':[1,2,3], 'c1':[4, 5, 6]}
df = pd.DataFrame(dict_data, index=['r0', 'r1', 'r2'])
print(df, "\n")

df.to_csv("df.csv")

df.to_excel("df.xlsx")
```

```
    c0  c1
r0   1   4
r1   2   5
r2   3   6
```